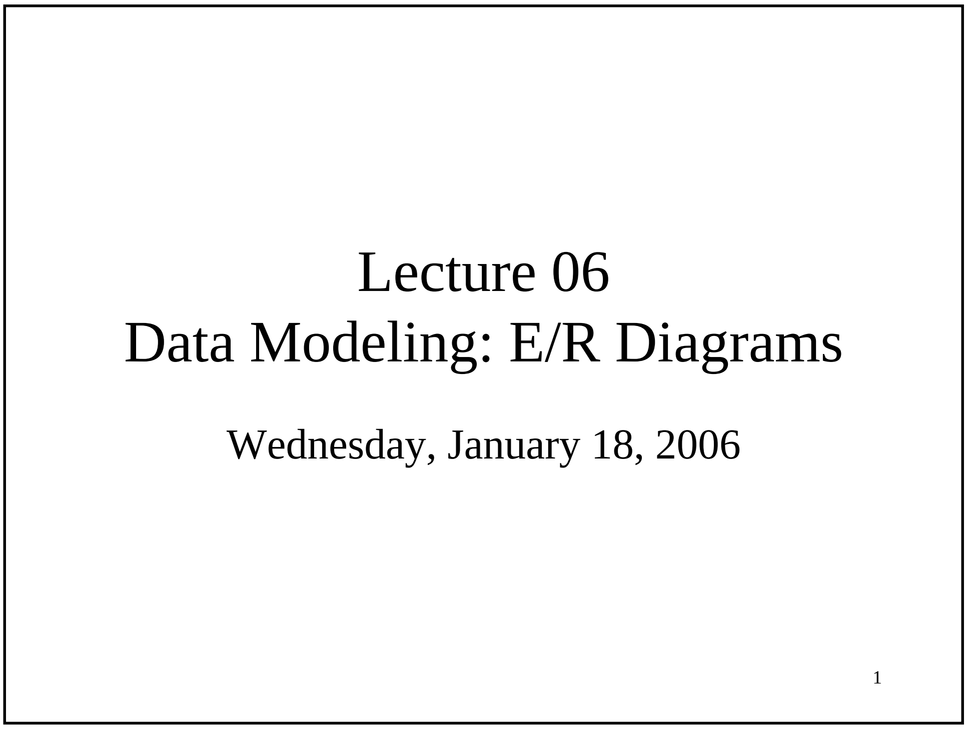
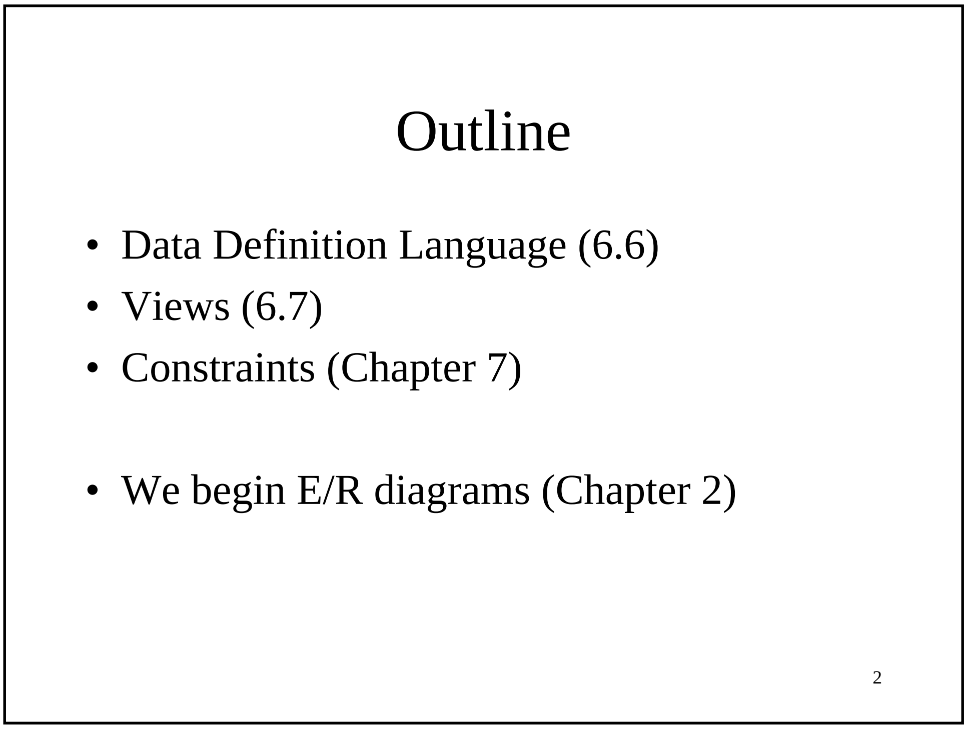


# Lecture 06
# Data Modeling: E/R Diagrams

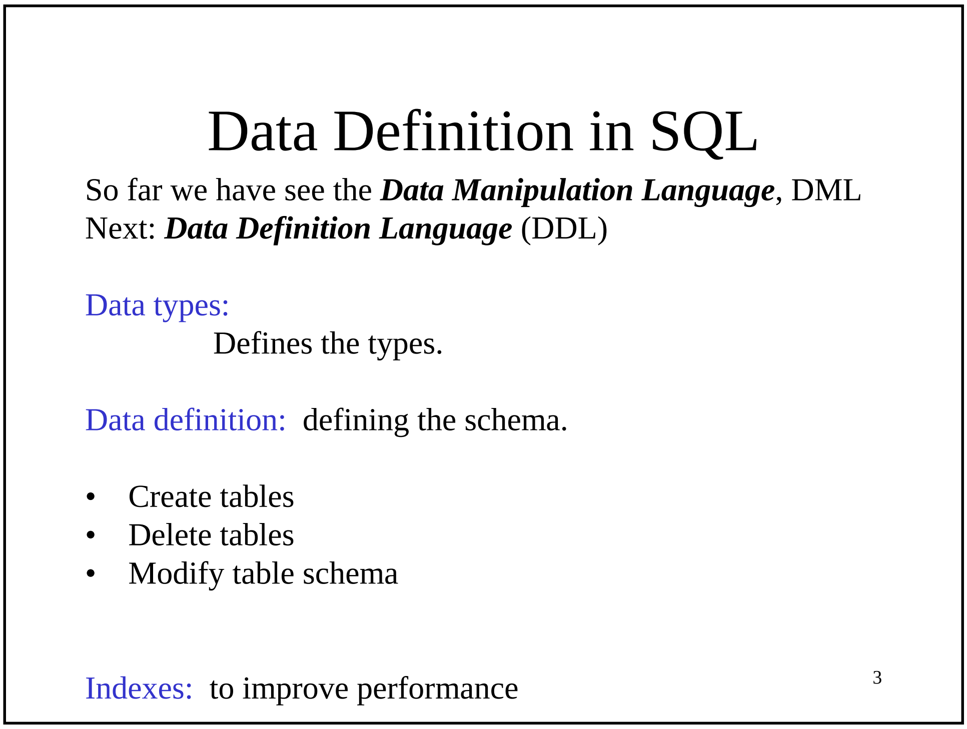
Wednesday, January 18, 2006



## Outline

- Data Definition Language (6.6)
- Views (6.7)
- Constraints (Chapter 7)

- We begin E/R diagrams (Chapter 2)



# Data Definition in SQL

So far we have see the ***Data Manipulation Language***, DML
Next: ***Data Definition Language*** (DDL)

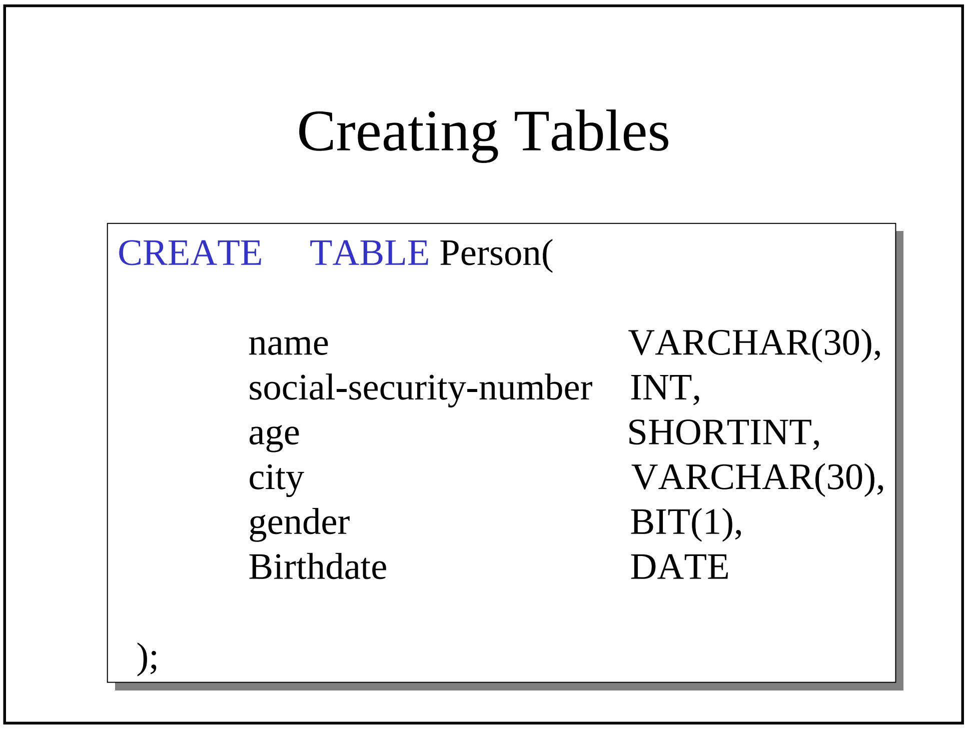
Data types:

Defines the types.

Data definition: defining the schema.

- Create tables
- Delete tables
- Modify table schema

Indexes: to improve performance



# Creating Tables

```
CREATE    TABLE Person(

          name                      VARCHAR(30),
          social-security-number    INT,
          age                       SHORTINT,
          city                      VARCHAR(30),
          gender                    BIT(1),
          Birthdate                 DATE

 );
```

# Deleting or Modifying a Table

Deleting:

Example:

```
DROP Person;
```

Exercise with care !!

Altering: (adding or removing an attribute).

Example:

```
ALTER TABLE   Person
          ADD   phone  CHAR(16);

ALTER  TABLE   Person
          DROP  age;
```

What happens when you make changes to the schema?

# Default Values

Specifying default values:

```
CREATE  TABLE Person(
          name                          VARCHAR(30),
          social-security-number  INT,
          age          SHORTINT   DEFAULT 100,
          city      VARCHAR(30) DEFAULT   ‘Seattle’,
          gender         CHAR(1)  DEFAULT  ‘?’,
          Birthdate                            DATE
```

The default of defaults:   NULL

# Indexes

**REALLY** important to speed up query processing time.

Suppose we have a relation

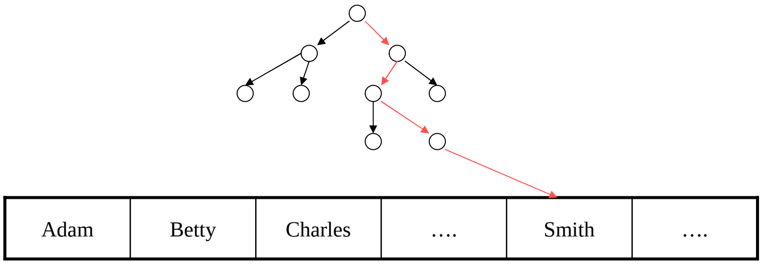
Person (name, age, city)

```
SELECT *
FROM    Person
WHERE   name = “Smith”
```

Sequential scan of the file Person may take long

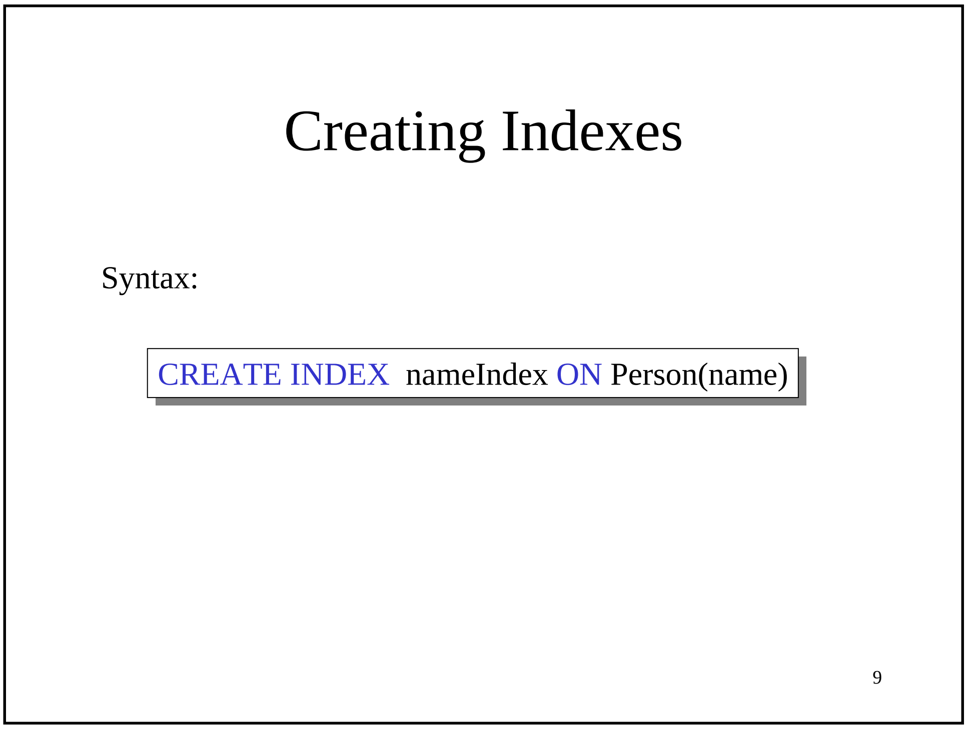
# Indexes

- Create an index on name:



| Adam | Betty | Charles | …. | Smith | …. |
|---|---|---|---|---|---|

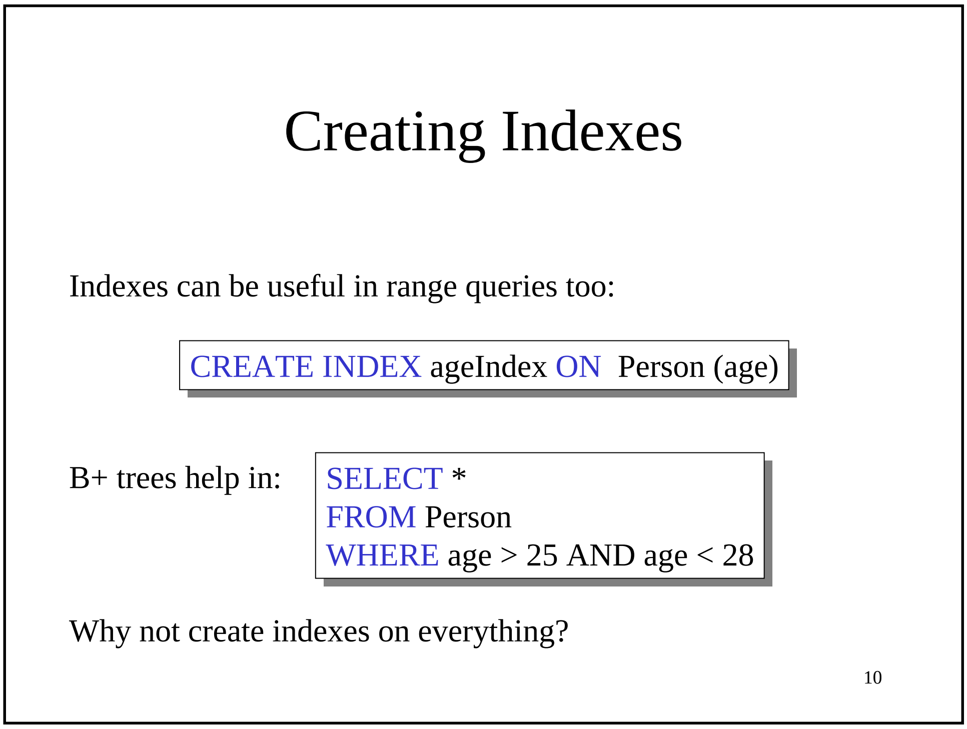
B+ trees have fan-out of 100s: max 4 levels !
Will discuss in the second half of this course



# Creating Indexes

Syntax:

```
CREATE INDEX  nameIndex ON Person(name)
```



# Creating Indexes

Indexes can be useful in range queries too:

```
CREATE INDEX ageIndex ON  Person (age)
```

B+ trees help in:

```
SELECT *
FROM Person
WHERE age > 25 AND age < 28
```

Why not create indexes on everything?

# Creating Indexes

Indexes can be created on more than one attribute:

Example:

```
CREATE INDEX doubleindex ON
                  Person (age, city)
```

Helps in:

```
SELECT *
FROM   Person
WHERE age = 55 AND city = “Seattle”
```

and even in:

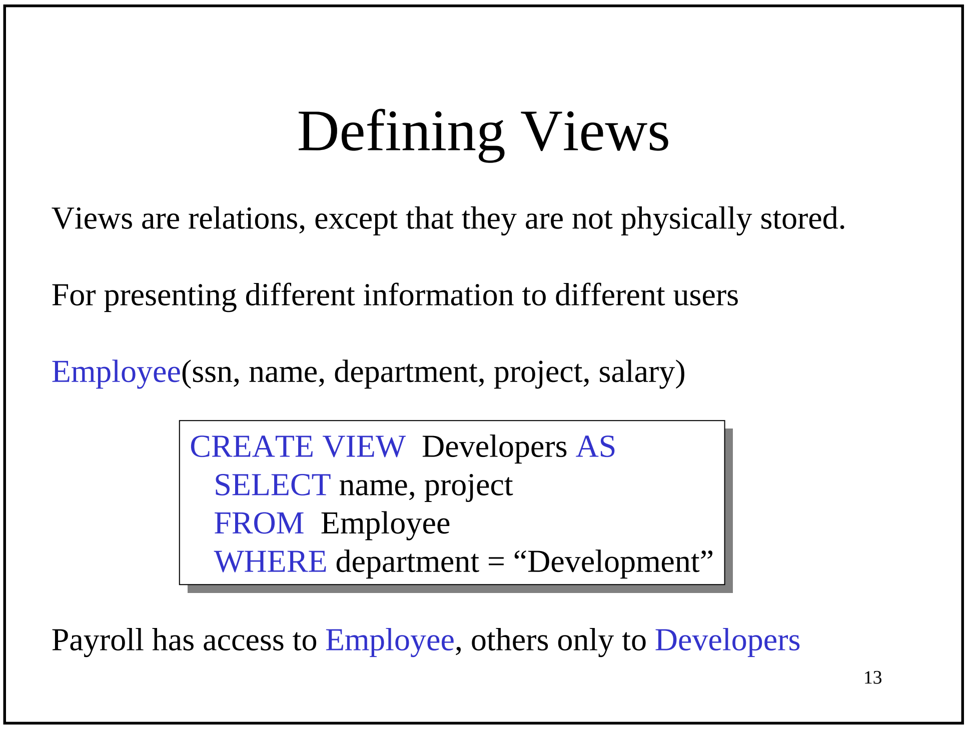
```
SELECT *
FROM   Person
WHERE age = 55
```

But not in:

```
SELECT *
FROM   Person
WHERE city = “Seattle”
```

# The Index Selection Problem

- Why not build an index on every attribute ? On every pair of attributes ?  Etc. ?
- The index selection problem is hard: balance the query cost v.s. the update cost, in a large application workload



# Defining Views

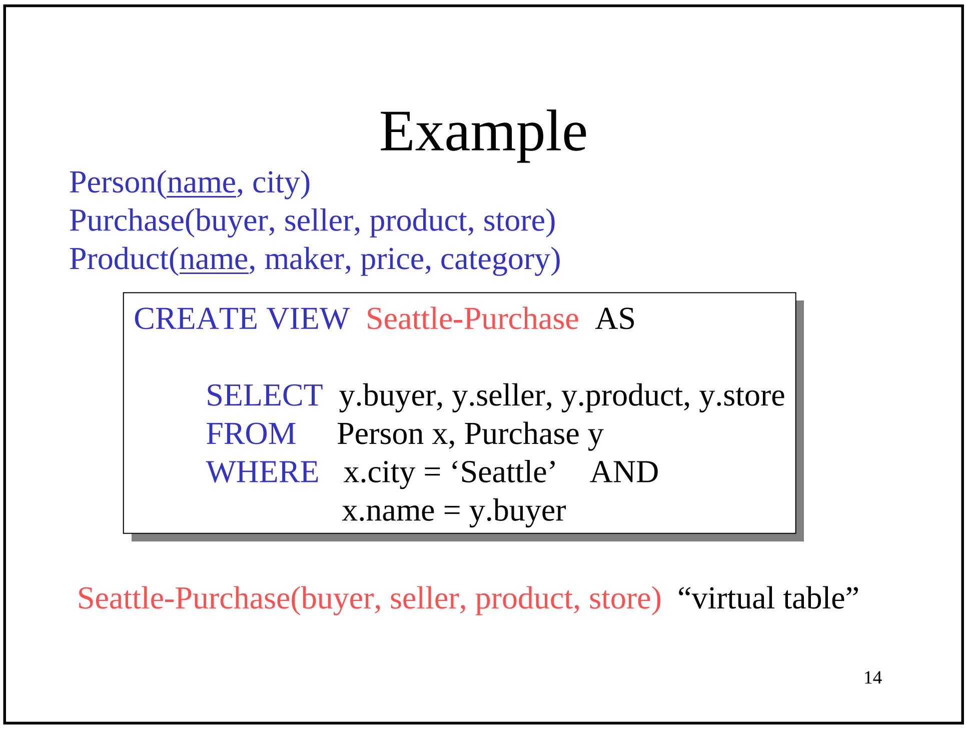
Views are relations, except that they are not physically stored.

For presenting different information to different users

Employee(ssn, name, department, project, salary)

```
CREATE VIEW  Developers AS
  SELECT name, project
  FROM  Employee
  WHERE department = “Development”
```

Payroll has access to Employee, others only to Developers



# Example

Person(name, city)
Purchase(buyer, seller, product, store)
Product(name, maker, price, category)

```
CREATE VIEW  Seattle-Purchase  AS

      SELECT  y.buyer, y.seller, y.product, y.store
      FROM     Person x, Purchase y
      WHERE   x.city = ‘Seattle’    AND
                 x.name = y.buyer
```

Seattle-Purchase(buyer, seller, product, store)  “virtual table”

We can later use the view:

```
SELECT  v.name, u.store
FROM    Seattle-Purchase u, Product v
WHERE   u.product = v.name  AND
        v.category = ‘shoes’
```

# What Happens When We Query a View ?

```
SELECT  v.name, u.store
FROM    Seattle-Purchase u, Product v
WHERE   u.product = v.name      AND
        v.category = ‘shoes’
```

↓

```
SELECT  v.name, y.store
FROM    Person x, Purchase y, Product v
WHERE   x.city = ‘Seattle’              AND
        x.name = y.buyer                AND
        y.product = v.name              AND
        v.category = ‘shoes’
```

# Types of Views

- Virtual views:
  - Used in databases
  - Computed only on-demand – slow at runtime
  - Always up to date
- Materialized views
  - Used in data warehouses
  - Pre-computed offline – fast at runtime
  - May have stale data

# Updating Views: Part 1

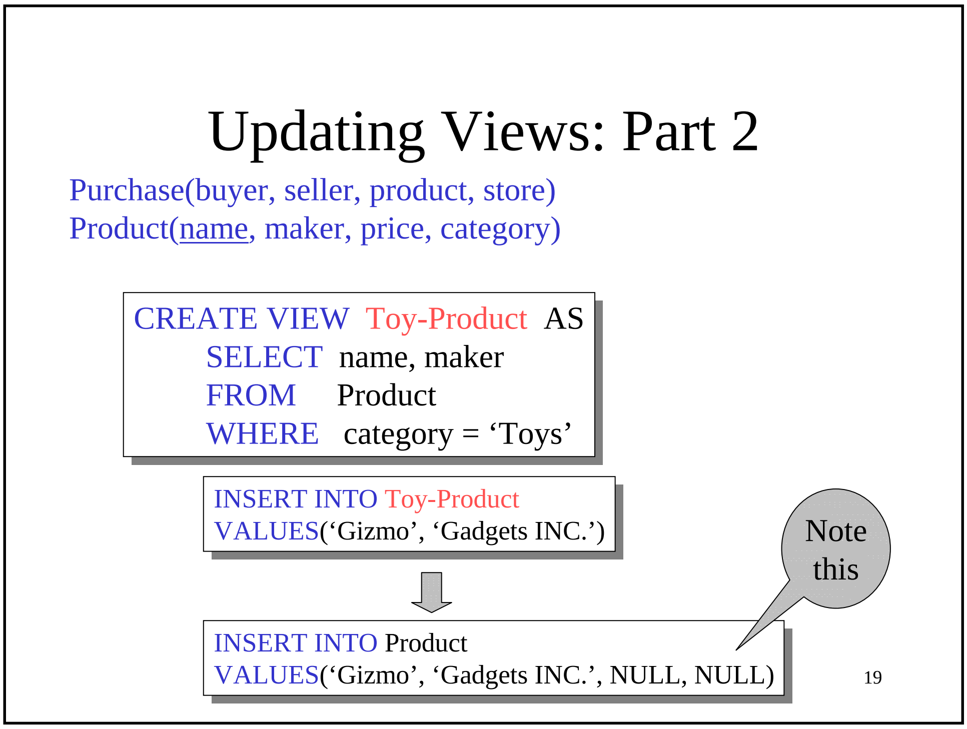
Purchase(buyer, seller, product, store)
Product(name, maker, price, category)

```
CREATE VIEW  Expensive-Product  AS
       SELECT  name, maker
       FROM     Product
       WHERE   price > 100
```

```
INSERT INTO Expensive-Product
VALUES('Gizmo', 'Gadgets INC.')
```

⇩

```
INSERT INTO Product
VALUES('Gizmo', 'Gadgets INC.', NULL, NULL)
```



# Updating Views: Part 2

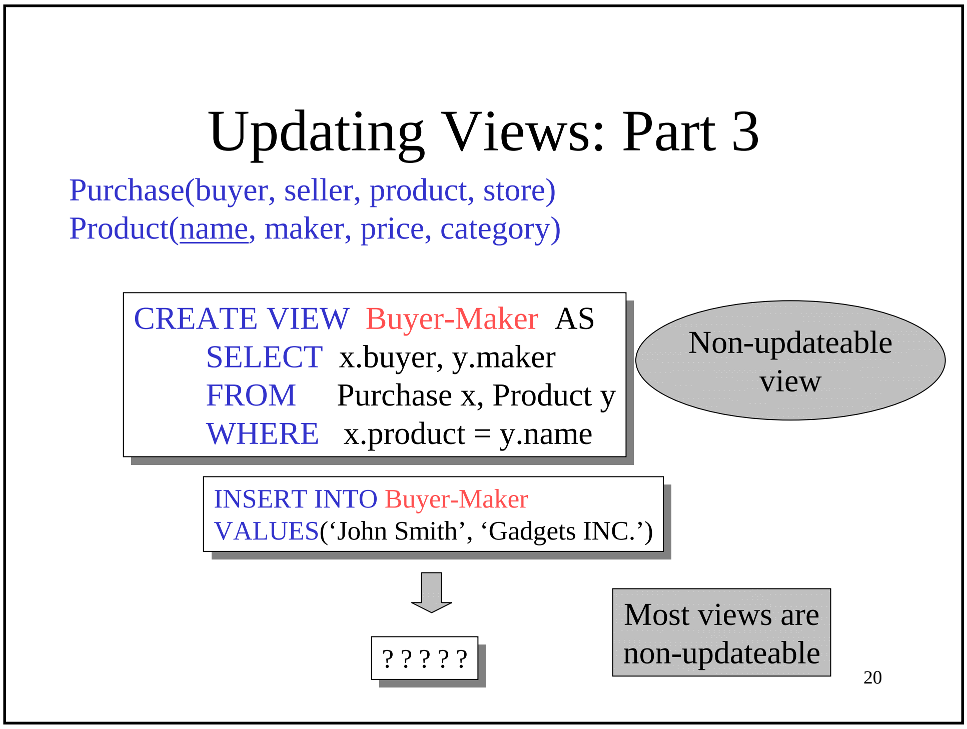
Purchase(buyer, seller, product, store)
Product(name, maker, price, category)

```
CREATE VIEW  Toy-Product  AS
        SELECT  name, maker
        FROM     Product
        WHERE   category = ‘Toys’
```

```
INSERT INTO Toy-Product
VALUES(‘Gizmo’, ‘Gadgets INC.’)
```

⇩

```
INSERT INTO Product
VALUES(‘Gizmo’, ‘Gadgets INC.’, NULL, NULL)
```



# Updating Views: Part 3

Purchase(buyer, seller, product, store)
Product(name, maker, price, category)

```
CREATE VIEW  Buyer-Maker  AS
        SELECT  x.buyer, y.maker
        FROM     Purchase x, Product y
        WHERE   x.product = y.name
```

Non-updateable view

```
INSERT INTO Buyer-Maker
VALUES(‘John Smith’, ‘Gadgets INC.’)
```

⇩

? ? ? ? ?

Most views are non-updateable

# Constraints in SQL

- A constraint = a property that we'd like our database to hold
- The system will enforce the constraint by taking some actions:
  - forbid an update
  - or perform compensating updates

# Constraints in SQL

Constraints in SQL:

- Keys, foreign keys
- Attribute-level constraints
- Tuple-level constraints
- Global constraints: assertions

simplest

Most complex

The more complex the constraint, the harder it is to check and to enforce

# Keys

```
CREATE TABLE Product (
          name CHAR(30) PRIMARY KEY,
          category VARCHAR(20))
```

OR:

Product(name, category)

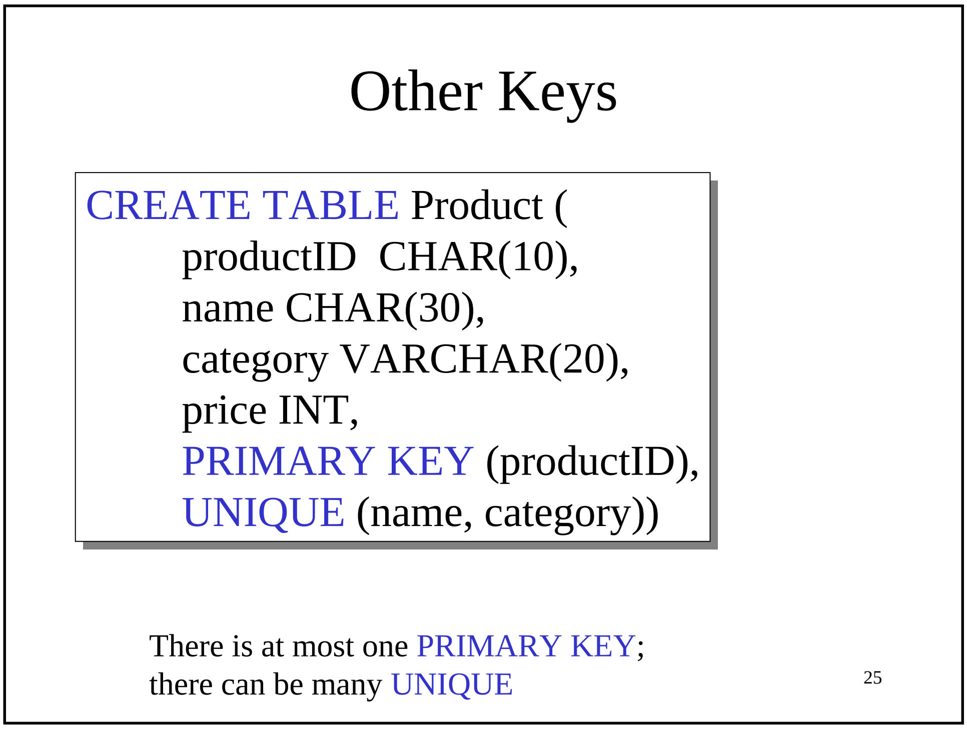
```
CREATE TABLE Product (
          name CHAR(30),
          category VARCHAR(20)
PRIMARY KEY (name))
```

# Keys with Multiple Attributes

```
CREATE TABLE Product (
          name CHAR(30),
          category VARCHAR(20),
          price INT,
     PRIMARY KEY (name, category))
```

| Name | Category | Price |
|---|---|---|
| Gizmo | Gadget | 10 |
| Camera | Photo | 20 |
| Gizmo | Photo | 30 |
| ~~Gizmo~~ | ~~Gadget~~ | ~~40~~ |

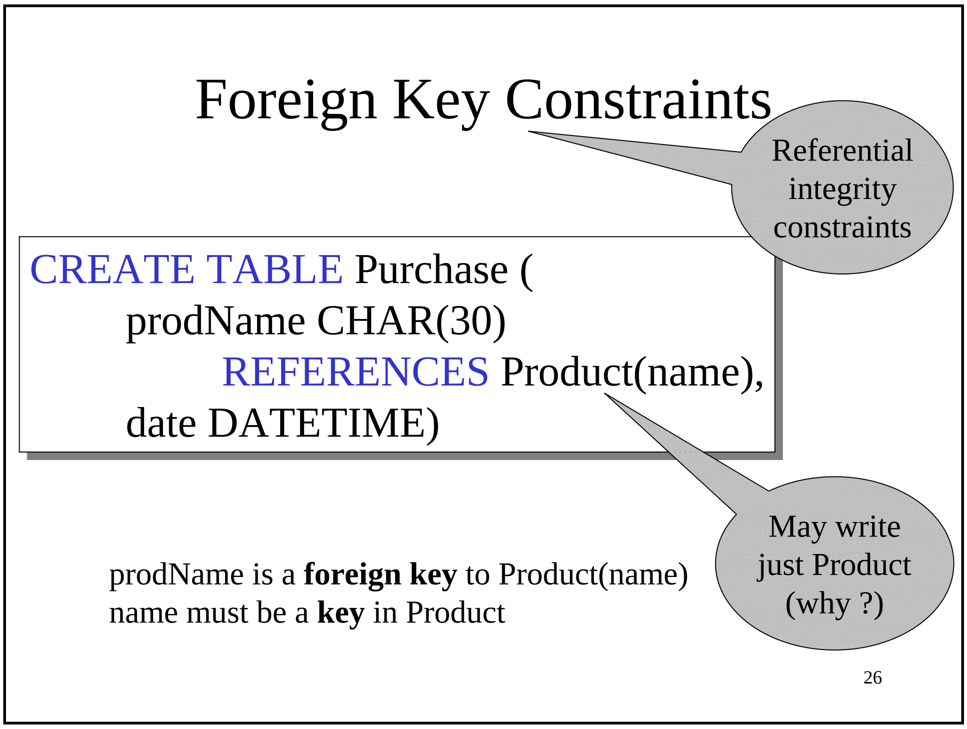
Product(name, category, price)



# Other Keys

```
CREATE TABLE Product (
        productID  CHAR(10),
        name CHAR(30),
        category VARCHAR(20),
        price INT,
        PRIMARY KEY (productID),
        UNIQUE (name, category))
```

There is at most one PRIMARY KEY;
there can be many UNIQUE



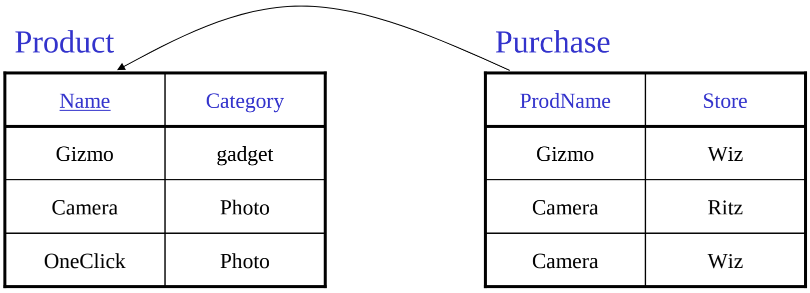
# Foreign Key Constraints

Referential integrity constraints

```
CREATE TABLE Purchase (
        prodName CHAR(30)
                REFERENCES Product(name),
        date DATETIME)
```

May write just Product (why ?)

prodName is a **foreign key** to Product(name)
name must be a **key** in Product



Product

| Name | Category |
|---|---|
| Gizmo | gadget |
| Camera | Photo |
| OneClick | Photo |

Purchase

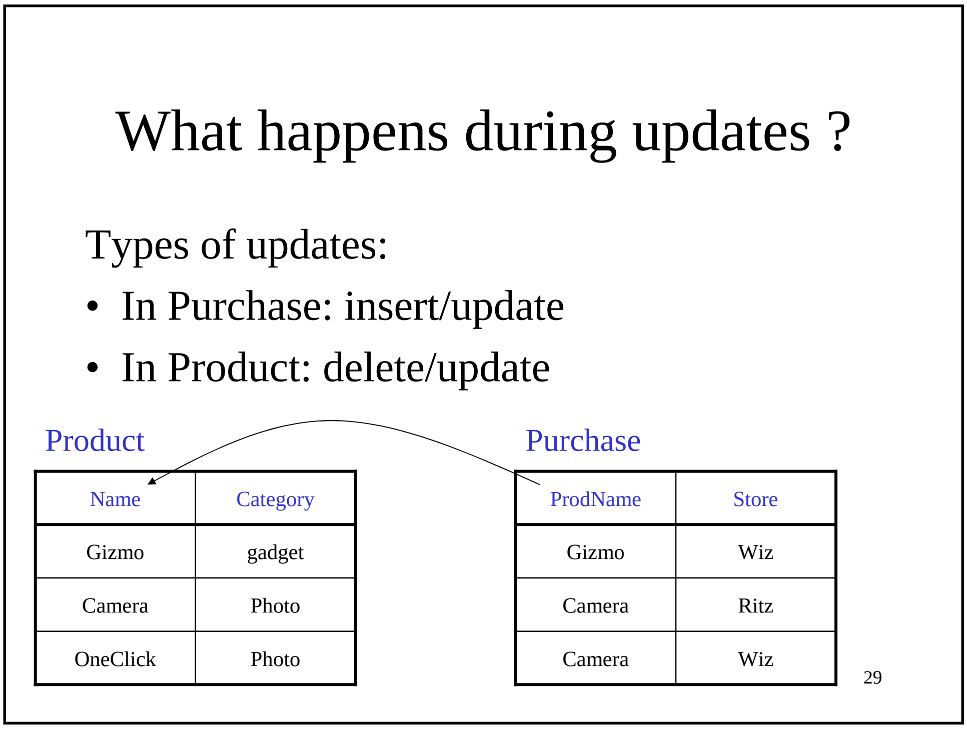
| ProdName | Store |
|---|---|
| Gizmo | Wiz |
| Camera | Ritz |
| Camera | Wiz |

# Foreign Key Constraints

- OR

```
CREATE TABLE Purchase (
        prodName CHAR(30),
        category VARCHAR(20),
        date DATETIME,
        FOREIGN KEY (prodName, category)
           REFERENCES  Product(name, category)
```

- (name, category) must be a PRIMARY KEY



## What happens during updates ?

Types of updates:

- In Purchase: insert/update
- In Product: delete/update

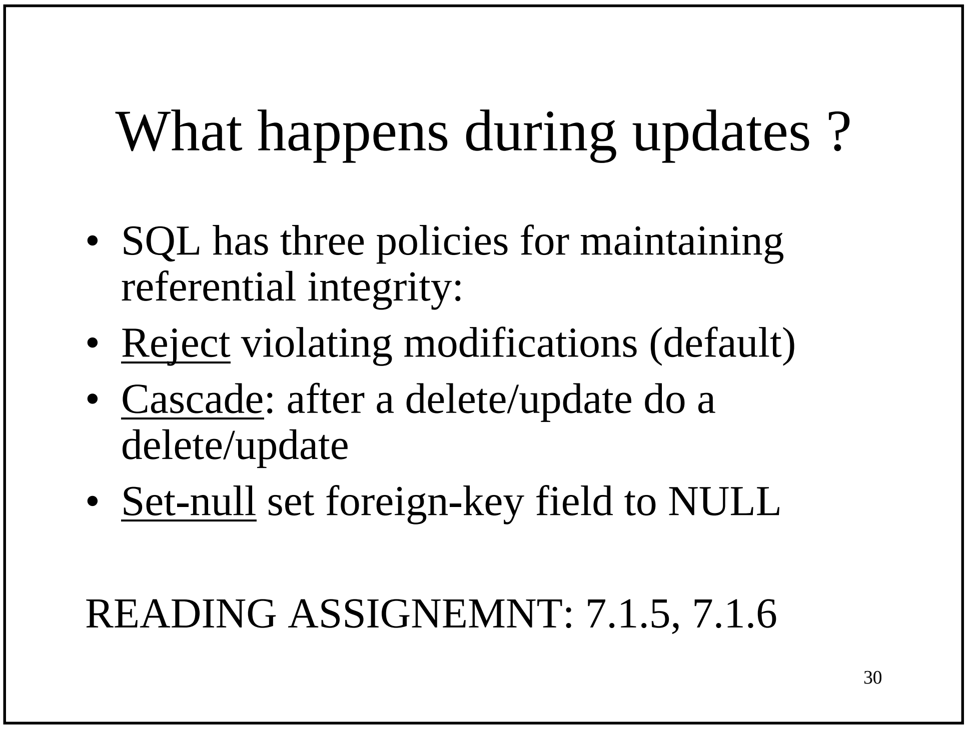
Product

| Name | Category |
|---|---|
| Gizmo | gadget |
| Camera | Photo |
| OneClick | Photo |

Purchase

| ProdName | Store |
|---|---|
| Gizmo | Wiz |
| Camera | Ritz |
| Camera | Wiz |

29

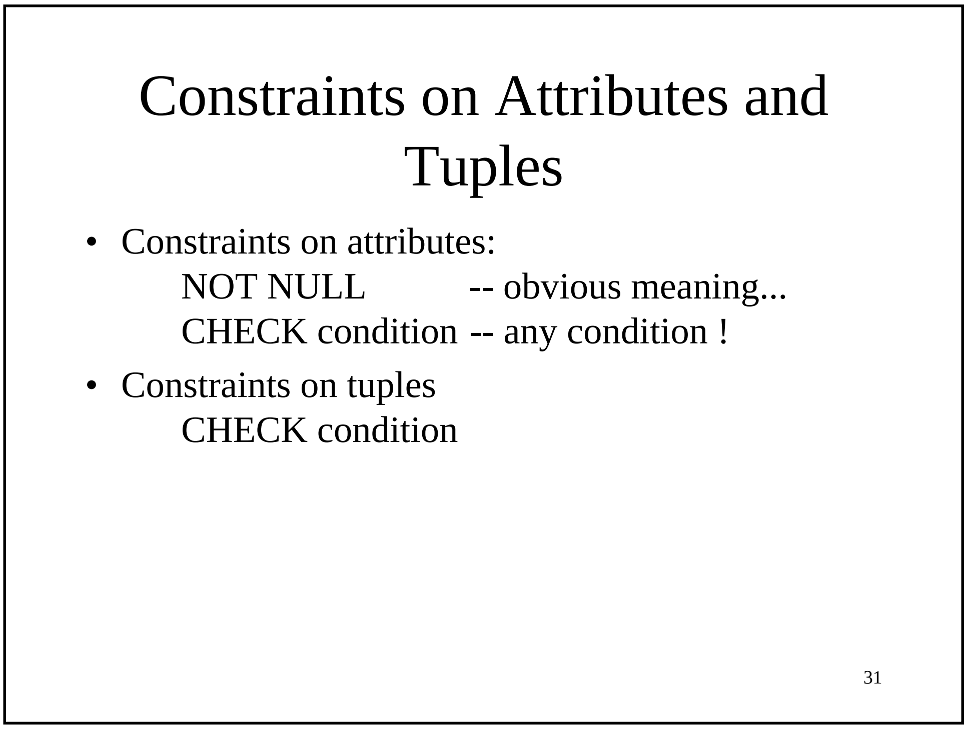


## What happens during updates ?

- SQL has three policies for maintaining referential integrity:
- Reject violating modifications (default)
- Cascade: after a delete/update do a delete/update
- Set-null set foreign-key field to NULL

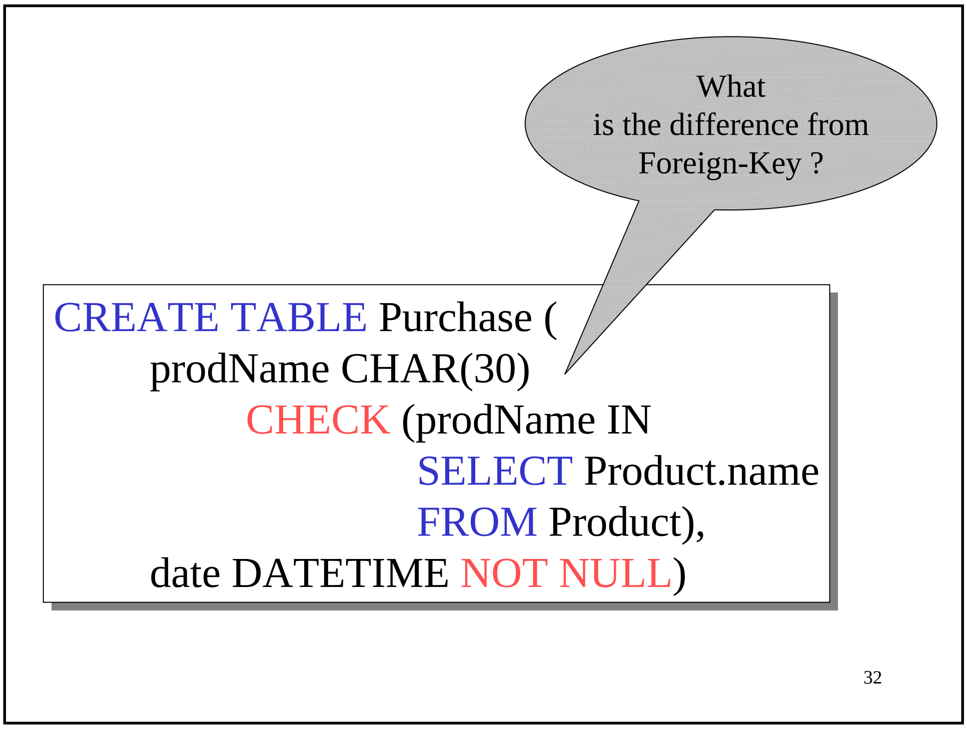
READING ASSIGNEMNT: 7.1.5, 7.1.6

30



# Constraints on Attributes and Tuples

- Constraints on attributes:
  NOT NULL -- obvious meaning...
  CHECK condition -- any condition !
- Constraints on tuples
  CHECK condition



What
is the difference from
Foreign-Key ?

```
CREATE TABLE Purchase (
        prodName CHAR(30)
                CHECK (prodName IN
                              SELECT Product.name
                              FROM Product),
        date DATETIME NOT NULL)
```

## General Assertions

```
CREATE ASSERTION myAssert CHECK
  NOT EXISTS(
         SELECT Product.name
         FROM Product, Purchase
         WHERE Product.name = Purchase.prodName
         GROUP BY Product.name
         HAVING count(*) > 200)
```

## Final Comments on Constraints

- Can give them names, and alter later
  - Read in the book !!!
- We need to understand exactly *when* they are checked
- We need to understand exactly *what* actions are taken if they fail